Introduction to Databases
《数据库引论》

Lecture 7: Relational Database Design Theory (2)
第7讲：关系数据库设计理论 (2)

周水庚 / Shuigeng Zhou

邮件: sgzhou@fudan.edu.cn 网址：admis.fudan.edu.cn/sgzhou

复旦大学计算机科学技术学院

Content of the Course

- **Part 0: Overview**
  - Lect. 0/1 (Feb. 20) - Ch1: Introduction
- **Part 1 Relational Databases**
  - Lect. 2 (Feb. 27) - Ch2: Relational model (data model, relational algebra)
  - Lect. 3 (Mar. 6) - Ch3: SQL (Introduction)
  - Lect. 4 (Mar. 13) – Ch4 & 5: Intermediate & Advanced SQL
- **Part 2 Database Design**
  - Lect. 5 (Mar. 20) - Ch6: Database design based on E-R model
  - Lect. 6 (Mar. 27) - Ch7: Relational database design (Part I)
  - Lect. 7 (Apr. 3) - Ch7: Relational database design (Part II)
- **Midterm exam: Apr. 10**
- **Part 3 Data Storage & Indexing**
  - Lect. 7 (Apr. 17) - Ch12/13: Storage systems & structures
  - Lect. 8 (Apr. 24) - Ch14: Indexing
- **Part 4 Query Processing & Optimization**
  - May 1, holiday, no classes
  - Lect. 9 (May 8) - Ch15: Query processing
  - Lect. 10 (May 15) - Ch16: Query optimization
- **Part 5 Transaction Management**
  - Lect. 11 (May 22) - Ch17: Transactions
  - Lect. 12 (May 29) - Ch18: Concurrency control
  - Lect. 13 (Jun. 5) - Ch19: Recovery system
  - Lect. 14 (Jun. 5) – Course review

Final exam: 13:00-15:00, Jun. 18

University Database

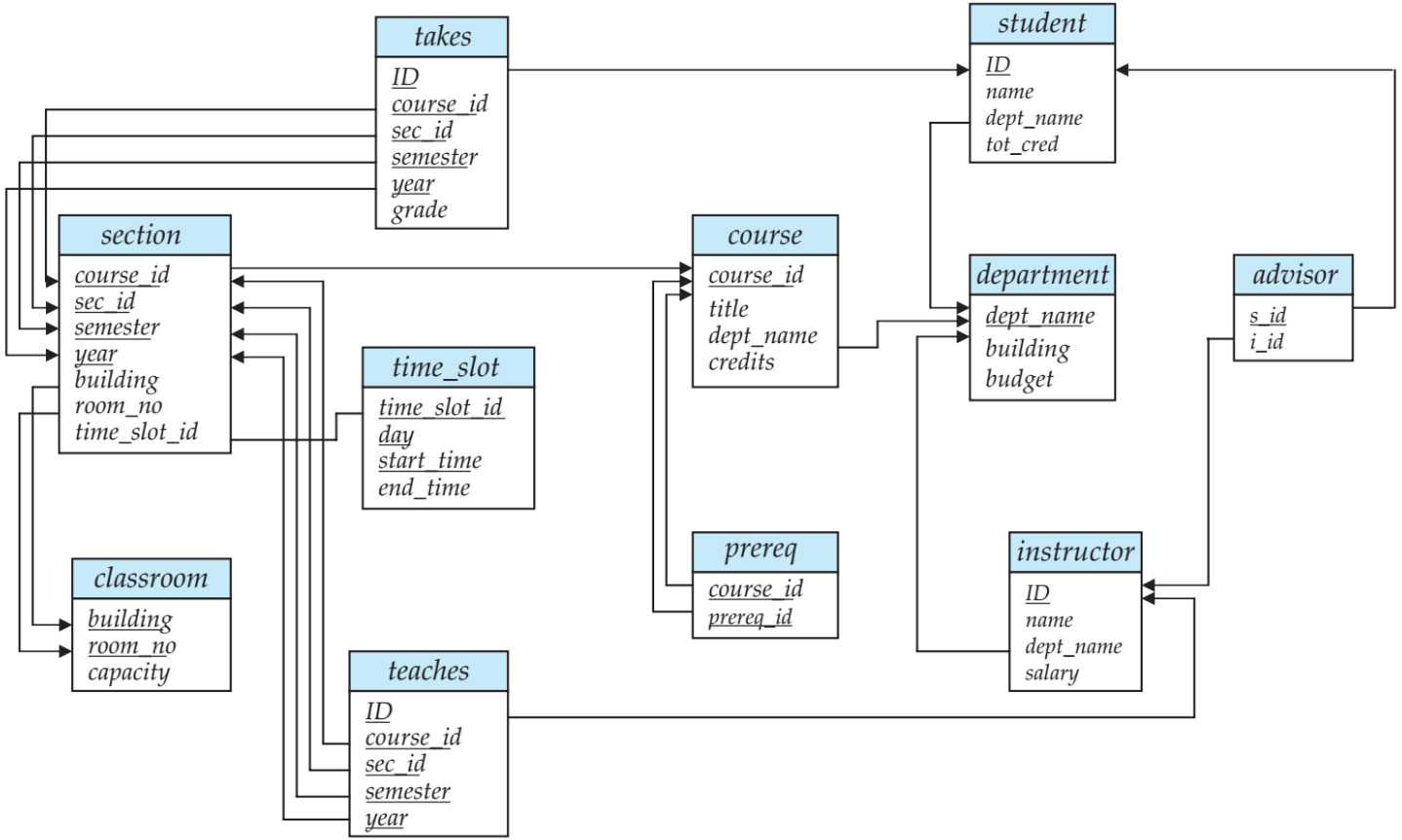
ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Instructor table

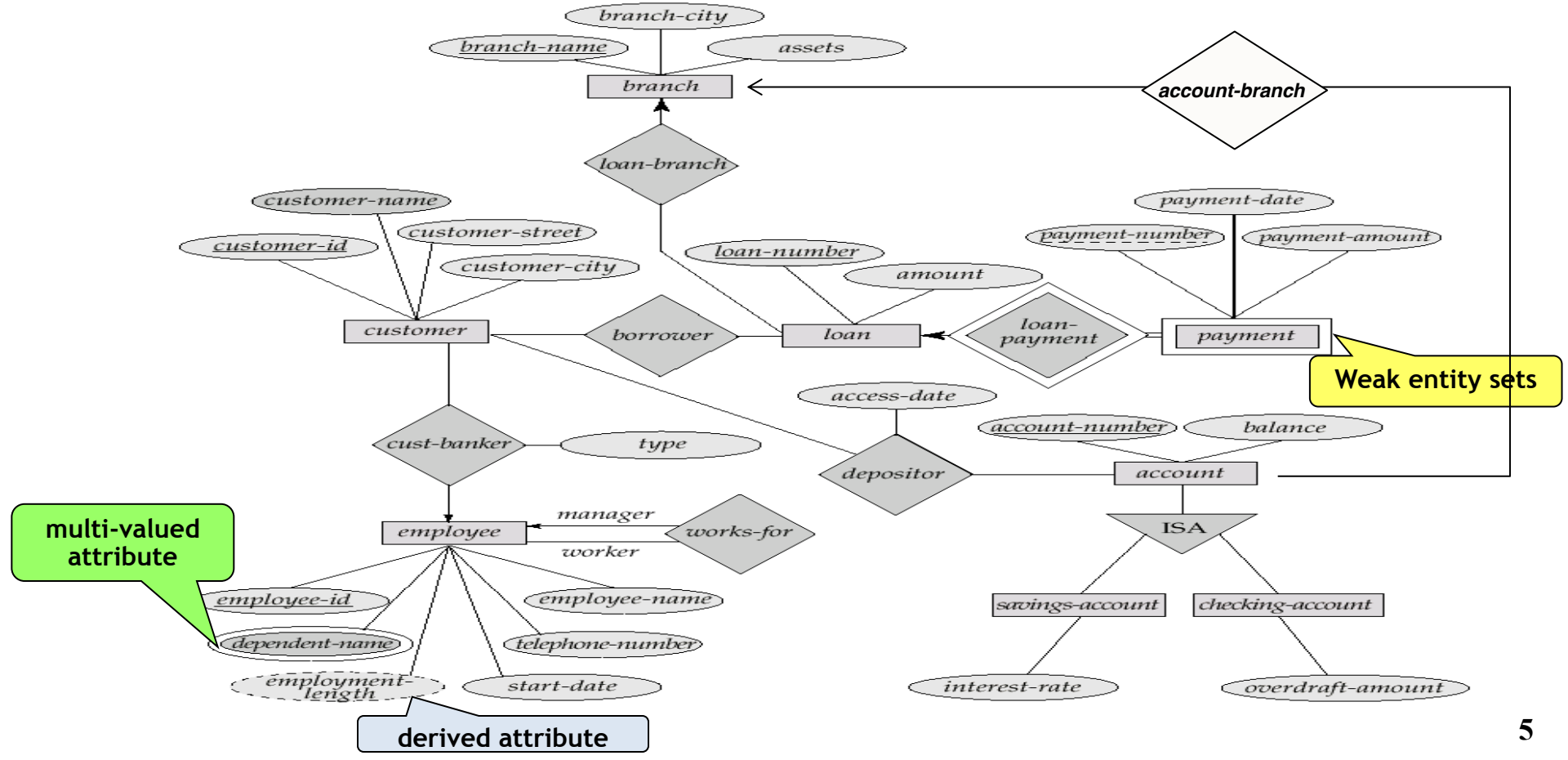
ID	*name*	*dept_name*	*tot_cred*
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

Student table

University Database



E-R Diagram for a Banking Enterprise



The Banking Schema

- *branch* = (*branch_name*, *branch_city*, *assets*)
- *customer* = (*customer_id*, *customer_name*, *customer_street*, *customer_city*)
- *loan* = (*loan_number*, *amount*)
- *account* = (*account_number*, *balance*)
- *employee* = (*employee_id*, *employee_name*, *telephone_number*, *start_date*)

- *dependent_name* = (*employee_id*, *dname*) (derived from a multivalued attribute)

- *account_branch* = (*account_number*, *branch_name*)
- *loan_branch* = (*loan_number*, *branch_name*)
- *cust_banker* = (*customer_id*, *employee_id*, *type*)
- *borrower* = (*customer_id*, *loan_number*)
- *depositor* = (*customer_id*, *account_number*, *access_date*)
- *works_for* = (*worker_employee_id*, *manager_employee_id*)

- *payment* =(*loan_number*,*payment_number*,*payment_date*,*payment_amount*)

- *savings_account* = (*account_number*, *interest_rate*)
- *checking_account* = (*account_number*, *overdraft_amount*)

Outline

- ☞ Normalization （规范化） & Normal Forms (范式)
- Multivalued Dependencies* （多值依赖)
- Database Design Process

Data Normalization (规范化)

- The process of decomposing relations with anomalies to produce smaller and well-structured relations

- To validate and improve a logical design so that it satisfies certain constraints that avoid unnecessary duplication of data

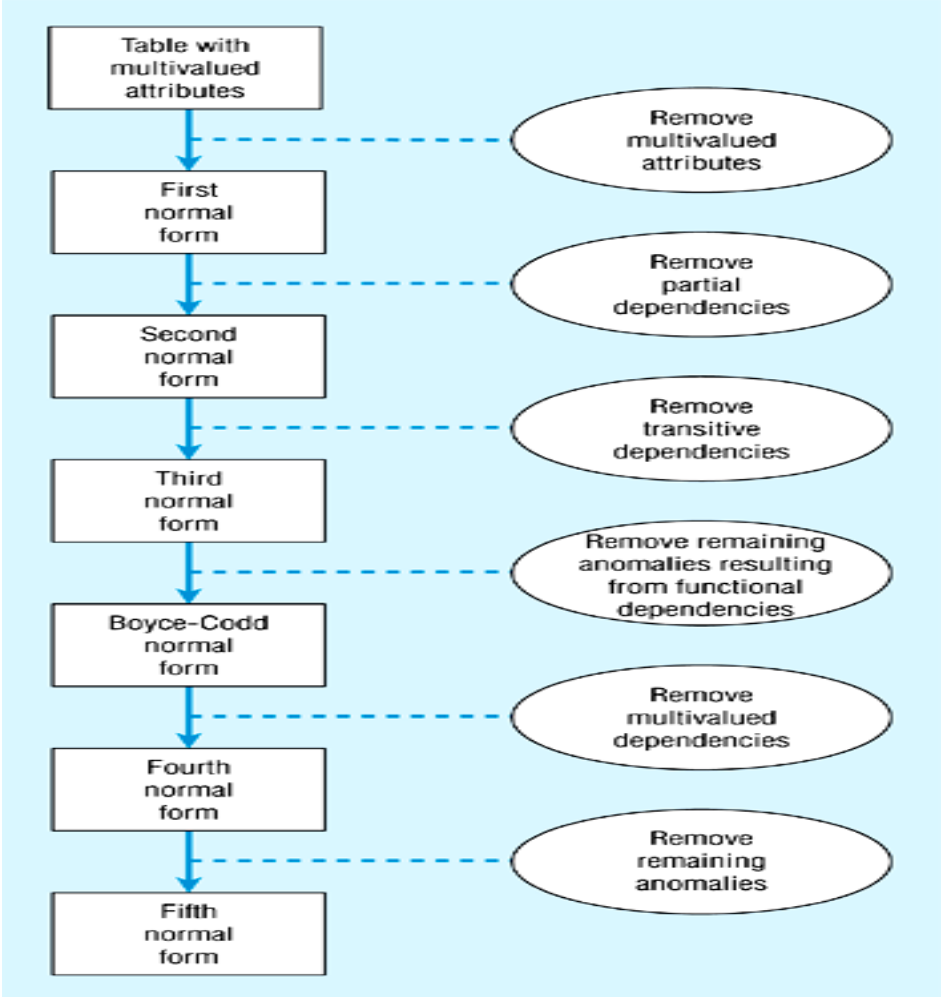
- The problems of having duplication of data
  - Waste of space
  - Difficulty in **consistency control**

Well-structured Relations

- A relation that contains minimal data redundancy and allows users to insert, delete, and update rows without causing data inconsistencies
- Goal is to avoid anomalies
  - **Insertion Anomaly** – adding new rows forces user to create duplicate data
  - **Deletion Anomaly** – deleting rows may cause a loss of data that would be needed for other future rows
  - **Modification Anomaly** – changing data in a row forces changes to other rows because of duplication

General rule of thumb: a table should not pertain to more than one entity type

Steps in Normalization



Atomic Domains and First Normal Form

- Domain is **atomic** if its elements are considered to be indivisible units
  - attributes do not have any substructure
- A relational **schema R is in 1NF** if the domains of all attributes of R are atomic
- Non-atomic values complicate storage and encourage redundant storage of data
  - E.g. composite attribute/ multivalued attributes

First Normal Form (1NF, Cont.)

- **Atomicity** is actually a property of how the elements of the domain are used
  - E.g. Strings would normally be considered indivisible
    - Suppose that students are given roll numbers which are strings of the form 0372001
  - If the first four characters are extracted to find the department, the domain of roll numbers is not atomic
    - Doing so is a bad idea: leads to encoding of information in application program rather than in the database

First Normal Form (1NF)

- Requirements
  - No multivalued attributes
    - Making each value a separate tuple (Not good idea!)
  - Every attribute value is atomic
    - Splitting a composite attribute into multiple non-composite attributes
- E.g.,
  - Fig. 1 is not in 1st Normal Form (multivalued attributes)
  - Fig. 2 is in 1st Normal form

- All relations should be in 1st Normal Form

Figure 1

not in 1NF (multivalued attributes)

Emp_ID	Name	Dept_Name	Salary	Course_Title	Date_Completed
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/200X
				Surveys	10/7/200X
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/200X
110	Chris Lucero	Info Systems	43,000	Visual Basic	1/12/200X
				C++	4/22/200X
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/16/200X
				Java	8/12/200X

Figure 2

in 1NF

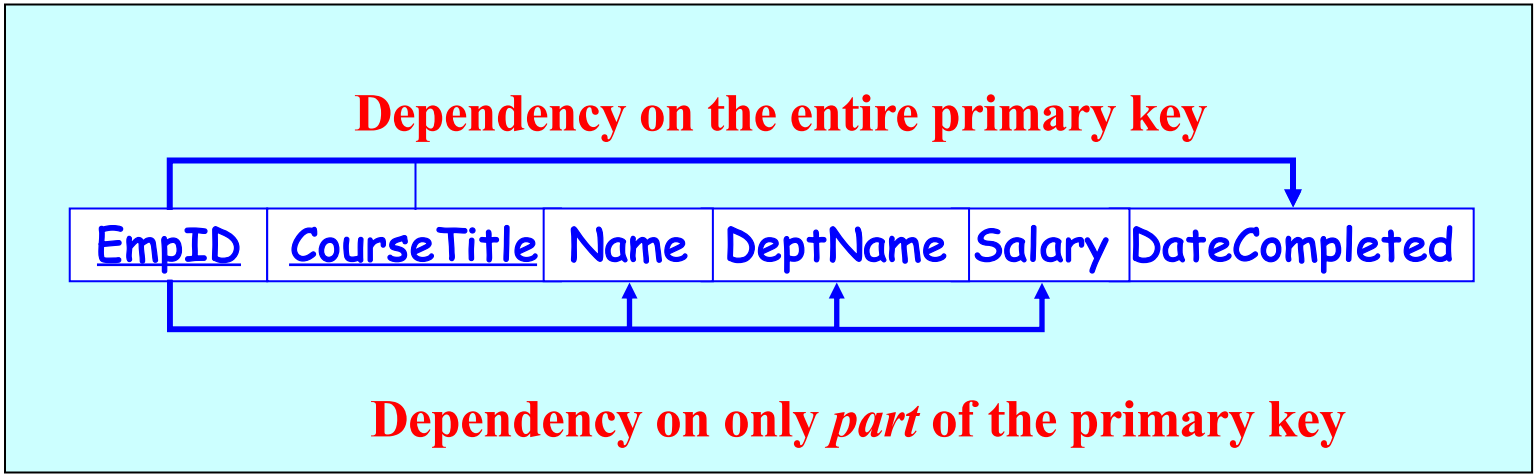
EMPLOYEE2

Emp_ID	Name	Dept_Name	Salary	Course_Title	Date_Completed
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/200X
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/200X
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/200X
110	Chris Lucero	Info Systems	43,000	Visual Basic	1/12/200X
110	Chris Lucero	Info Systems	43,000	C++	4/22/200X
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/200X
150	Susan Martin	Marketing	42,000	Java	8/12/200X

Second Normal Form

- 2nd Normal Form
  - 1NF
  - Every non-key attribute is fully functionally dependent on the ENTIRE primary key, i.e., no partial functional dependencies
- Partial functional dependency
  - A function dependency in which one or more non-key attributes are functionally dependent on part (but not in all) of the primary key

Functional Dependencies in Employee



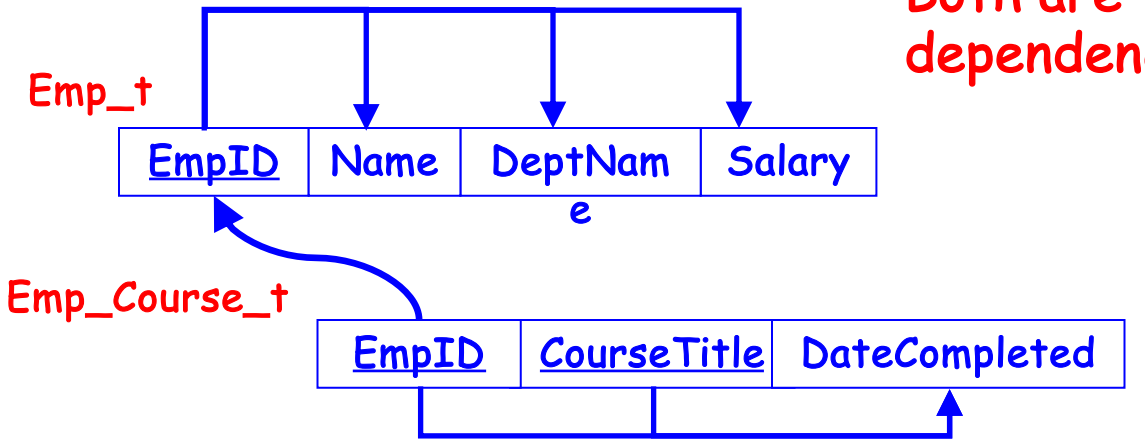
EmpID, CourseTitle ➔ DateCompleted

EmpID ➔ Name, DeptName, Salary

As such, NOT in 2nd Normal Form!

Decompose a Relation to 2nd Normal Form

- Decompose the relation into two separate relations



Both are full functional dependencies

Third Normal Form

- **Requirements**
  - 2NF
  - No transitive dependencies
- A **transitive dependency** is a functional dependency between two (or more) non-key attributes.

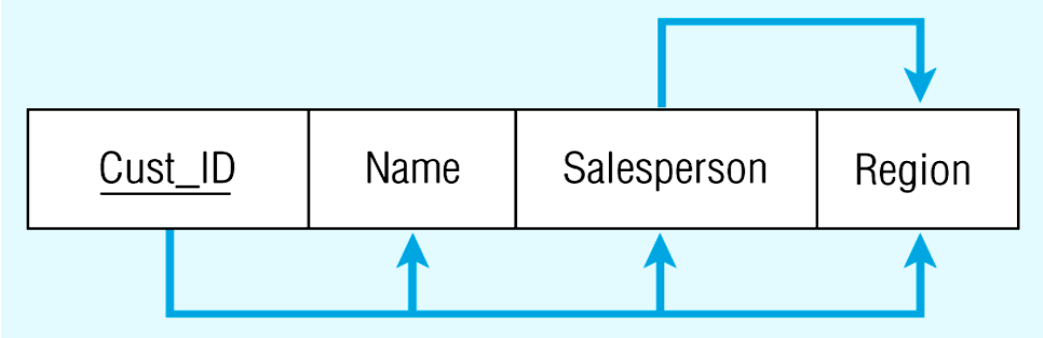
Relation with Transitive Dependency

SALES

Cust_ID	Name	Salesperson	Region
8023	Anderson	Smith	South
9167	Bancroft	Hicks	West
7924	Hobbs	Smith	South
6837	Tucker	Hernandez	East
8596	Eckersley	Hicks	West
7018	Arnold	Faulb	North

SALES relation

Relation with Transitive Dependency



Cust_ID → Name
Cust_ID → Salesperson
Cust_ID → Region

All this is OK
(2nd NF)

BUT

Cust_ID → Salesperson → Region

Transitive dependency
(not 3rd NF)

Relation with Transitive Dependency

SALES1

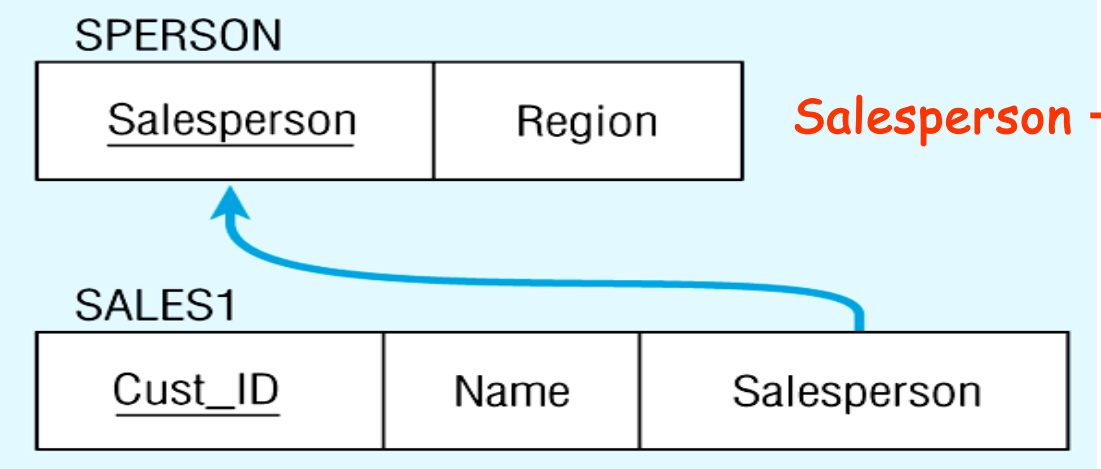
Cust_ID	Name	Salesperson
8023	Anderson	Smith
9167	Bancroft	Hicks
7924	Hobbs	Smith
6837	Tucker	Hernandez
8596	Eckersley	Hicks
7018	Arnold	Faulb

SPERSON

Salesperson	Region
Smith	South
Hicks	West
Hernandez	East
Faulb	North

Decompose the SALES relation

Relations in 3NF



Salesperson → Region

Cust_ID → Name

Cust_ID → Salesperson

Now, there are no transitive dependencies…
Both relations are in 3rd NF

Data Normalization

- 1st Normal Form
  - No multivalued attributes, and every attribute value is atomic
  - All relations are in 1st Normal Form
- 2nd Normal Form
  - 1NF + every non-key attribute is fully functionally dependent on the ENTIRE primary key
  - Decomposing the relation into two new relations
- 3rd Normal Form
  - 2NF + no transitive dependencies
  - Decomposing the relation into two new relations

Other Normal Forms

- **Boyce-Codd NF**
  - **All determinants are superkeys**
- **4th NF**
  - No multivalued dependencies
- **5th NF**
  - Join dependencies generalize MVDs
    - Lead to the project-join normal form (PJNF), or the 5th NF
- A class of even more general constraints, leads to a normal form called **domain-key normal form**
- Problem with these generalized constraints: are hard to reason with, and no set of sound and complete set of inference rules exists

Boyce-Codd Normal Form

- Given relation schema R and FDs F, **R is BCNF** if for every FD $\alpha \rightarrow \beta$ in F^+ ($\alpha \subseteq R$ and $\beta \subseteq R$), **at least** one of the following holds:
  - $\alpha \rightarrow \beta$ **is trivial** (i.e., $\beta \subseteq \alpha$)
  - α **is a superkey for R**

Example

- $R = (A, B, C)$, $F = \{A \rightarrow B, B \rightarrow C\}$, Key = $\{A\}$
  - R is not in BCNF since $B \rightarrow C$ but B is not the key

- Decomposition $R_1 = (A, B)$, $R_2 = (B, C)$
  - R_1 and R_2 in BCNF
  - Lossless-join decomposition
  - Dependency preserving

Testing for BCNF

- To check if a non-trivial dependency $\alpha \rightarrow \beta$ in F^+ causes a violation of BCNF
  - compute α^+ (the attribute closure of α), and
  - verify that it includes all attributes of R, i.e., a superkey of R

- Simplified test
  - To check if a relation schema R is in BCNF, it suffices to check only the FDs F for violation of BCNF, rather than checking all dependencies in F^+
  - If none of the dependencies in F causes a violation of BCNF, then none of the dependencies in F^+ will cause a violation of BCNF either

Testing for BCNF (Cont.)

- Using only F is **incorrect** when testing a relation in a **decomposition** of **R**
- E.g., consider **R (A, B, C, D)** with **F = {A → B, B → C}**
  - Decompose **R** into **R_1(A,B)** and **R_2(A,C,D)**
  - Neither of the dependencies in **F** contain only attributes from **(A,C,D)** so we might be mislead into thinking that **R_2** satisfies **BCNF**
  - In fact, **dependency A → C in F^+** shows that **R_2 is not in BCNF**

Testing Decomposition for BCNF

- To check if a relation R_i in a decomposition of R is in BCNF
  - Either test R_i for BCNF w.r.t. the restriction of F to R_i (that is, all FDs in F^+ that contain only attributes from R_i)
  - or use the original set of dependencies F that hold on R, but with the following test:
    - for every set of attributes $\alpha \subseteq R_i$, check that α^+ either includes no attributes of $R_i - \alpha$（要么不是决定属性）, or includes all attributes of R_i （要么是R_i超键）.
    - If the condition is violated by some $\alpha \rightarrow \beta$ in F, the FD $\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$ can be shown to hold on R_i, and R_i violates BCNF
    - We use above dependency to decompose R_i

BCNF Decomposition Algorithm

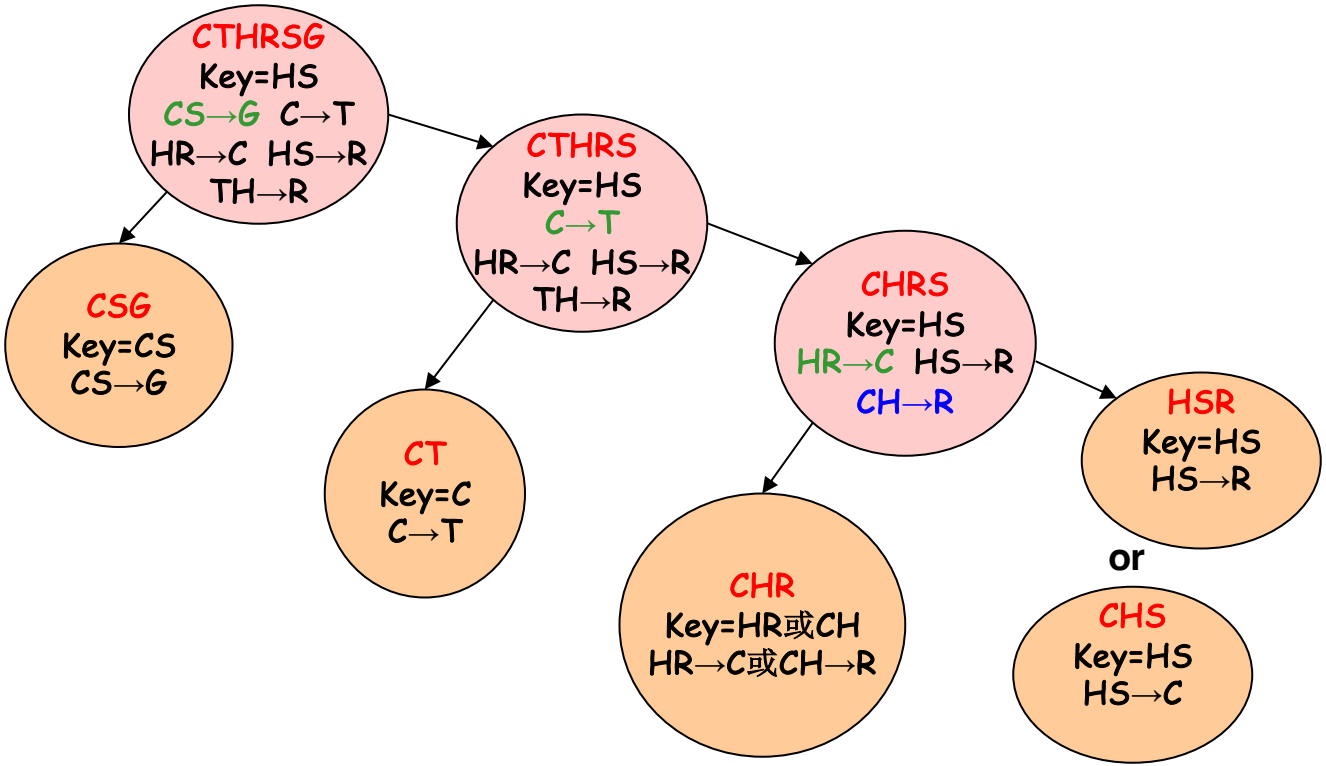
```
result := {R};
done := false;
while (not done) do
    if (there is a schema R_i in result that is not in BCNF)
      then begin
              let α → β be a nontrivial functional dependency that holds
              on R_i such that α+ does not contain R_i and α ∩ β = ∅;
              result := (result − R_i) ∪ (R_i − β) ∪ ( α, β);
           end
      else done := true;
```

Note: each R_i is in BCNF, and decomposition is lossless-join

Example

- Consider the relation scheme **CTHRSG**, where **C**=course, **T**=teacher, **H**=hour, **R**=room, **S**=student, and **G**=grade. The functional dependencies **F** we assume are:
  - **CS→G**: each student has one grade in each course
  - **C→T**: each course has one teacher
  - **HR→C**: only one course can meet in a room at one time
  - **HS→R**: a student can be in only one room at one time
  - **TH→R**: a teacher can be in only one room at one time

Decomposition Tree



BCNF and Dependency Preservation

It is not always possible to get a BCNF decomposition that is dependency preserving

- R = (J, K, L), F = {JK → L, L → K}, two candidate keys = JK and JL
  - R is not in BCNF
- Any decomposition of R will fail to preserve
  - JK → L 或者 L → K

Third Normal Form: Motivation

- There are some situations where
  - BCNF is not dependency preserving, and
  - Efficient checking for FD violation on updates is important

- **Solution:** define a weaker normal form, i.e., **Third Normal Form**
  - Allows some redundancy
  - But FDs can be checked on individual relations without computing a join
  - **There is always a lossless-join, dependency-preserving decomposition into 3NF**

Third Normal Form

- A relation schema R is in 3NF if for all $\alpha \rightarrow \beta$ in F^+ at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
  - α is a superkey for R
  - Each attribute A in $\beta - \alpha$ is contained in a candidate key for R

  (NOTE: each attribute may be in a different candidate key)

- If a relation is in BCNF, it is in 3NF (since in BCNF one of the first two conditions above must hold)
- Third condition is a minimal relaxation of BCNF to ensure dependency preservation

3NF (Cont.)

- **Example**
  - R = (J, K, L), F = {JK → L, L → K}, two candidate keys: JK and JL
  - **R is in 3NF**
    - JK → L JK is a superkey/ candidate key
    - L → K K is contained in a candidate key
  - BCNF decomposition has (JL) and (LK), and testing for JK → L requires a join
- There is some redundancy in this schema
- Equivalent to example:

  Banker-schema = (branch-name, customer-name, banker-name)

  banker-name → branch name,

  branch-name, customer-name → banker-name

Testing for 3NF

- **Optimization:** Need to check only FDs in F
- Use **attribute closure** to check for each dependency $\alpha \rightarrow \beta$, if α is a **superkey**.
- If α is **not a superkey**, we have to **verify if each attribute in β is contained in a candidate key of R**
  - this test is rather more expensive, since it involve finding candidate keys
  - testing for **3NF** has been shown to be NP-hard
  - Interestingly, **decomposition into 3NF can be done in polynomial time**

3NF Decomposition Algorithm

Let F_c be a canonical cover for F;
$i := 0$;
for each FD $\alpha \rightarrow \beta$ in F_c do
if none of the schemas R_j, $1 \le j \le i$ contains α, β
then begin
$i := i + 1$;
$R_i := \alpha \beta$
end

end
if none of the schemas R_j, $1 \le j \le i$ contains a candidate key for R
then begin
$i := i + 1$;
R_i := any candidate key for R;
end
return $(R_1, R_2, \ldots, R_i)$

The algorithm ensures that each relation schema R_i is in 3NF, and decomposition is dependency preserving and lossless-join

3NF Decomposition Example 1

- R<U, F>, U={A,B,C,D,E}, F={AB→CDE, AC→BDE, B→C, C→D, B→E}
  - R is in which NF? Decompose R into 3NF, and the decomposition is dependency preserving and lossless-join

1) Fc={AC→B, B→CE, C→D};

2) Find candidate keys: AC、AB;
   - key-attributes are: A、B、C;
   - for C→D, non-key attribute D is partial dependent on key AC, so R ∉ 2NF, R∈1NF.

3) Decompose R into 3NF:
   - So decompose R into (Same LHS attributes):
     - U1={A,B,C}, F1={AC → B }
     - U2={B,C,E}, F2={B → CE }
     - U3={C,D}, F3={C → D }
   - ρ={R1<U1,F1>, R2<U2,F2>, R3<U3,F3>}, the decomposition is dependency preserving. And candidate keys AC、AB are all in U1, so a row can be found as a1, a2, a3, a4, a5 for testing lossless-join form, so ρ is lossless-join.

3NF Decomposition Example 2

- R<U, F>, U={A,B,C,D}, F={A→C, C→A, B→AC, D→AC, BD→A}.
  - R is in which NF? Decompose R into 3NF, and the decomposition is dependency preserving and lossless-join

  1) Fc={A → C, C → A, B → A, D → A}

  2) Candidate keys of R: BD; key-attributes: B、D;
     - For B→A and D→A, non-key attribute A is partial dependent on key BD, so R∉2NF, R∈ 1NF

  3) Decompose R into 3NF:
     - All attributes exist in F, and does not exist X→A ∈ F and XA=U
     - So decompose R into (Same LHS attributes):
       - U1={A,C}, F1={ A → C, C → A }
       - U2={A,B}, F2={ B → A }
       - U3={A,D}, F3={ D → A }
     - ρ={R1<U1,F1>, R2<U2,F2>, R3<U3,F3>}, the decomposition is dependency preserving. But candidate key BD is not in any Ui, so τ =ρ ∪ {R*<X,Fx>} = ρ ∪ {R4<{B,D},Φ>}, and τ is the decomposition that is dependency preserving and lossless-join
     - (ABCD)->(AC), (ABD) -> (AC), (AB), (AD), (BD)

Example

- Relation schema:

  Banker-info-schema = (branch-name, customer-name, banker-name, office-number)

- The FDs for this relation schema are:

  banker-name → branch-name, office-number
  customer-name, branch-name → banker-name

- The key is:

  {customer-name, branch-name}

Applying 3NF to *Banker-info-schema*

- The for loop in the algorithm causes us to include the following schemas in our decomposition:

  Banker-office-schema = (banker-name, branch-name, office-number)
  Banker-schema = (customer-name, branch-name, banker-name)

- Since Banker-schema contains a candidate key for Banker-info-schema, we are done with the decomposition process

- (*branch-name, customer-name, banker-name, office-number*) =⟫ (*banker-name, branch-name, office-number*) ∪ (*customer-name, branch-name, banker-name*)

Comparison of BCNF and 3NF

- It is always possible to decompose a relation into relations in 3NF and
  - the decomposition is lossless
  - the dependencies are preserved

- It is always possible to decompose a relation into relations in BCNF and
  - the decomposition is lossless
  - it may not be possible to preserve dependencies.

Comparison of BCNF and 3NF (Cont.)

- Example of problems due to redundancy in 3NF
  - R = (J, K, L)
    F = {JK → L, L → K}

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
null	l_2	k_2

- A schema that is in 3NF but not in BCNF has the problems of repetition of
  - information (e.g., the relationship l_1, k_1)
  - need to use null values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J)

Design Goals

- Goal for a relational database design:
  - BCNF
  - Lossless join
  - Dependency preservation

- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF

Design Goals (Cont.)

- Interestingly, SQL does not provide a direct way of specifying FDs other than superkeys.
  - Can specify FDs using assertions, but they are expensive to test

- Even if we had a dependency preserving decomposition, using SQL we would **not** be able to efficiently test a FD whose left hand side is not a key.

Testing for FDs Across Relations

- If decomposition is not dependency preserving, we can have an extra materialized view for each dependency $\alpha \rightarrow \beta$ in F_c that is not preserved in the decomposition
- The materialized view is defined as a projection on $\alpha\beta$ of the join of the relations in the decomposition
- Many newer database systems support materialized views and database system maintains the view when the relations are updated.
  - No extra coding effort for programmer

Testing for FDs Across Relations (Cont.)

- The functional dependency $\alpha \rightarrow \beta$ is expressed by declaring α as a candidate key on the materialized view
- Checking for candidate key cheaper than checking $\alpha \rightarrow \beta$
- **BUT:**
  - **Space overhead**: for storing the materialized view
  - **Time overhead**: Need to keep materialized view up to date when relations are updated
  - Database system may not support key declarations on materialized views

Outline

- Normalization （规范化） & Normal Forms (范式)

☞ Multivalued Dependencies* （多值依赖）

- Database Design Process

Multivalued Dependencies

- There are database schemas in **BCNF** that do not seem to be sufficiently normalized

- Consider a database

  classes(course, teacher, book)

  such that **(c, t, b)∈ classes** means that ***t*** is qualified to teach ***c***, and ***b*** is a required textbook for ***c***

- The database is supposed to list for each course the set of teachers any one of which can be the course's instructor, and the set of books, all of which are required for the course

course	*teacher*	*book*
database	Avi	DB Concepts
database	Avi	Ullman
database	Hank	DB Concepts
database	Hank	Ullman
database	Sudarshan	DB Concepts
database	Sudarshan	Ullman
operating systems	Avi	OS Concepts
operating systems	Avi	Shaw
operating systems	Jim	OS Concepts
operating systems	Jim	Shaw

classes

- There are no non-trivial functional dependencies and therefore the relation is in **BCNF**
- **Insertion anomalies** – i.e., if Sara is a new teacher that can teach database, two tuples need to be inserted
  - (database, Sara, DB Concepts)
  - (database, Sara, Ullman)

Multivalued Dependencies (Cont.)

- Therefore, it is better to decompose classes into:

course	*teacher*
database	Avi
database	Hank
database	Sudarshan
operating systems	Avi
operating systems	Jim

teaches

We shall see that these two relations are in 4NF

course	*book*
database	DB Concepts
database	Ullman
operating systems	OS Concepts
operating systems	Shaw

text

Multivalued Dependencies (MVDs)

- Let R be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$. The multivalued dependency

$$\alpha \twoheadrightarrow \beta$$

holds on R if in any legal relation r(R), for all pairs for tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, there exist tuples t_3 and t_4 in r such that:

$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$
$t_3[\beta] = t_1[\beta]$
$t_3[R - \beta] = t_2[R - \beta]$
$t_4[\beta] = t_2[\beta]$
$t_4[R - \beta] = t_1[R - \beta]$

- Why called "multivalued dependency"?
  - because a value of α determine multiple values of β

Why Called Multivalued Dependencies?

- When we say $\alpha \twoheadrightarrow \beta$, it means that a value of α determine multiple values of β

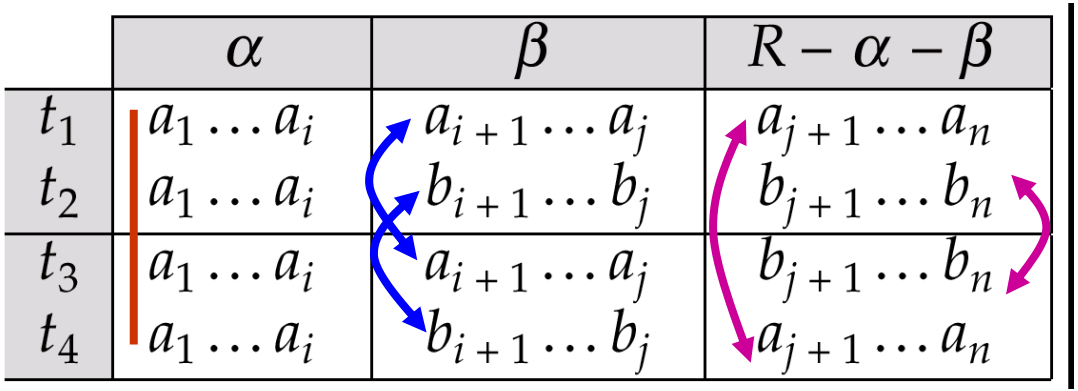
course	teacher	book
database	Avi	DB Concepts
database	Avi	Ullman
database	Hank	DB Concepts
database	Hank	Ullman
database	Sudarshan	DB Concepts
database	Sudarshan	Ullman
operating systems	Avi	OS Concepts
operating systems	Avi	Shaw
operating systems	Jim	OS Concepts
operating systems	Jim	Shaw

classes

We have: course $\twoheadrightarrow$ teacher, course $\twoheadrightarrow$ book

MVD (Cont.)

Tabular representation of $\alpha \twoheadrightarrow \beta$



Functional dependencies: equality-generating dependencies 相等产生依赖

Multivalued dependencies: tuple-generating dependencies 元组产生依赖

MVD (Cont.)

- Properties of MVD
  - Symmetry: if $X \twoheadrightarrow Y$ then $X \twoheadrightarrow Z$, here $Z=U-X-Y$
  - Transitivity: if $X \twoheadrightarrow Y$, $Y \twoheadrightarrow Z$, then $X \twoheadrightarrow Z-Y$
  - If $X \twoheadrightarrow Y$, $X \twoheadrightarrow Z$, then $X \twoheadrightarrow YZ$
  - If $X \twoheadrightarrow Y$, $X \twoheadrightarrow Z$, then $X \twoheadrightarrow Y \cap Z$
  - If $X \twoheadrightarrow Y$, $X \twoheadrightarrow Z$, then $X \twoheadrightarrow Y-Z$, $X \twoheadrightarrow Z-Y$
  - …

Example

- Let **R** be a relation schema with a set of attributes that are partitioned into 3 nonempty subsets.

  Y, Z, W

- We say that **Y↠Z** (**Y multi-determines Z**)
  iff for all possible relations **r(R)**
  - $< y, z_1, w_1 > \in r$ and $< y, z_2, w_2 > \in r$ then
  - $< y, z_1, w_2 > \in r$ and $< y, z_2, w_1 > \in r$
- Note that since the behavior of **Z** and **W** are identical it follows that **Y↠ Z** if **Y↠W**

Example (Cont.)

- In our example:
  - course ↠ teacher
  - course ↠ book
- The above formal definition is supposed to formalize the notion that given a particular value of **Y (course)** it has associated with it a set of values of **Z (teacher)** and a set of values of **W (book)**, and these two sets are in some sense independent of each other
- Note:
  - If **Y → Z** then **Y ↠ Z**
  - Indeed we have (in above notation) $z_1 = z_2$
    The claim follows

Use of Multivalued Dependencies

- We use **MVDs** in two ways:
  - 1. **To test relations** to determine whether they are legal under a given set of FDs and MVDs
  - 2. **To specify constraints** on the set of legal relations. We shall concern ourselves with relations that satisfy a given set of FDs and MVDs.
- If a relation r fails to satisfy a given MVD, we can construct a relations r' that does satisfy the MVD by adding tuples to r

Theory of MVDs

- From the definition of multivalued dependency, we can derive the following rule:
  - **If $\alpha \rightarrow \beta$, then $\alpha \twoheadrightarrow \beta$**; That is, every FD is also a MVD
- The closure **D^+** of D is the set of all FDs and MVDs logically implied by D.
- We can compute **D^+** from D, using the formal definitions of FDs and MVDs.
- We can manage with such reasoning for very simple MVDs, which seem to be common in practice
- For complex MVDs, it is better to reason about sets of dependencies using a system of inference rules

Fourth Normal Form

- A relation schema **R** is in **4NF** w.r.t. a set **D** of FDs and MVDs if for all MVDs in D^+ of the form $\alpha \twoheadrightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following hold:
  - $\alpha \twoheadrightarrow \beta$ **is trivial** (i.e., $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$)
  - α is a superkey for schema **R**
- If a relation is in **4NF** it is in **BCNF**

Restriction of Multivalued Dependencies

- The restriction of D to R_i is the set D_i consisting of
  - All FDs in D^+ that include only attributes of Ri
  - All MVDs of the form

    $$\alpha \twoheadrightarrow \beta \cap R_i$$

    where $\alpha \subseteq R_i$ and $\alpha \twoheadrightarrow \beta$ is in D^+

4NF Decomposition Algorithm

result: = {R};
done := false;
compute D+;
Let D_i denote the restriction of D+ to R_i
while (not done)
  if (there is a schema R_i in result that is not in 4NF) then
    begin
      let $\alpha \twoheadrightarrow \beta$ be a nontrivial MVD that holds on R_i such that $\alpha \rightarrow R_i$
      is not in D_i, and $\alpha \cap \beta = \emptyset$;
      result := (result - R_i) $\cup$ ((R_i - β) $\cup$ (α, β));
    end
  else done:= true;

Note: each R_i is in 4NF, and decomposition is lossless-join

Example

- $R = (A, B, C, G, H, I)$

  $F = \{ A \twoheadrightarrow B$

  $B \twoheadrightarrow HI$

  $CG \twoheadrightarrow H \}$

- R is not in 4NF since $A \twoheadrightarrow B$ and A is not a superkey for R
- Decomposition

a) $R_1 = (A, B)$ (R_1 is in 4NF)

b) $R_2 = (A, C, G, H, I)$ (R_2 is not in 4NF)

c) $R_3 = (C, G, H)$ (R_3 is in 4NF)

d) $R_4 = (A, C, G, I)$ (R_4 is not in 4NF)

- Since $A \twoheadrightarrow B$ and $B \twoheadrightarrow HI$, $A \twoheadrightarrow HI$, $A \twoheadrightarrow I$

e) $R_5 = (A, I)$ (R_5 is in 4NF)

f) $R_6 = (A, C, G)$ (R_6 is in 4NF)

Further Normal Forms

- Join dependencies generalize MVDs
  - lead to **project-join normal form (PJNF)** (also called **fifth normal form**)
    投影-连接范式

- A class of even more general constraints, leads to a normal form called domain-key normal form (DKNF) 域-码范式

- Problem with these generalized constraints: are hard to reason with, and no set of sound and complete set of inference rules exists, hence rarely used

Outline

- Normalization （规范化） & Normal Forms (范式)
- Multivalued Dependencies* （多值依赖)

☞ Database Design Process

Overall Database Design Process

- We have assumed schema **R** is given
  - **R** could have been generated when converting E-R diagram to a set of tables
  - **R** could have been a single relation containing all attributes that are of interest (called universal relation, 泛关系)
  - Normalization breaks **R** into smaller relations and normal form

ER Model and Normalization

- When an **E-R diagram** is carefully designed, identifying all **entities** correctly, the **tables** generated from the E-R diagram should not need further normalization
- However, in a real (imperfect) design there can be FDs from non-key attributes of an entity to other attributes of the entity
- **E.g.** employee entity with attributes department-number and department-address, and an FD

  department-number → department-address

  - Good design would have made department an entity
- FDs from non-key attributes of a relationship set are possible, but rare

Universal Relation Approach 泛关系

- **Dangling tuples（悬浮元组）** – Tuples that "disappear" in computing a join
  - Let $r_1(R_1), r_2(R_2), \ldots, r_n(R_n)$ be a set of relations
  - A tuple t of the relation r_i is a dangling tuple if t is not in the relation:

$$\Pi_{R_i} \bowtie (r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n)$$

- The relation $r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n$ is called a universal relation since it involves all the attributes in the "universe" defined by $R_1 \cup R_2 \cup \cdots \cup R_n$
- If dangling tuples are allowed in the database, instead of decomposing a universal relation, we may prefer to synthesize a collection of normal form schemas from a given set of attributes.

Universal Relation Approach

- Dangling tuples may occur in practical database applications
- They represent incomplete information
- E.g., may want to break up information about loans into:
  - (branch-name, loan-number)
  - (loan-number, amount)
  - (loan-number, customer-name)
- Universal relation would require null values, and have dangling tuples

Universal Relation Approach (Cont.)

- A particular decomposition defines a restricted form of incomplete information that is acceptable in our database.
  - Above decomposition requires at least one of customer-name, branch-name or amount in order to enter a loan number without using null values
  - Rules out storing of customer-name, amount without an appropriate loan-number (since it is a key, it can't be null either!)
- **Universal relation requires unique attribute names unique role assumption**
- Reuse of attribute names is natural in SQL since relation names can be prefixed to disambiguate names

Denormalization for Performance

- May want to use **non-normalized schema** for performance
  - E.g., displaying customer-name along with account-number and balance requires join of account with depositor
  - **Alternative 1: Use denormalized relation** containing attributes of account as well as depositor with all above attributes
    - Faster lookup
    - Extra space and extra execution time for updates
    - Extra coding work for programmer and possibility of error in extra code
  - **Alternative 2: use a materialized view** defined as
    account ⋈ depositor
    - Benefits and drawbacks same as above, except no extra coding work for programmer and avoids possible errors

Other Design Issues

- Some aspects of database design are not caught by normalization
- Examples of bad database design to be avoided: Instead of earnings(company-id, year, amount), use
  - earnings-2000, earnings-2001, earnings-2002, etc., all on the schema (company-id, earnings).
    - Above are in BCNF, but make querying across years difficult and needs a new table each year
  - company-year(company-id, earnings-2000, earnings-2001, earnings-2002)
    - Also in BCNF, but makes querying across years difficult and requires new attribute each year.
    - Is an example of a crosstab (交叉表), where values for one attribute become column names
    - Used in spreadsheets, and in data analysis tools

Quiz

- **Given the relational schema R<U, F>, U={A,B,C,D,E }, F={AC→BD, B → C, C → D, B → E}**
  a) Use Armstrong axioms and related rules to prove the functional dependency $AC \rightarrow E$
  b) Compute $(A)^+$ and $(AC)^+$
  c) Find a canonical cover F_c of F
  d) Find all candidate keys, and point out R is in which normal form
  e) Decompose R into 3NF, which the decomposition is lossless-join and dependency preserving.
  f) Give related explanation or proof that the above decomposition is lossless-join and dependency preserving
  g) *Decompose the relation into relations in BCNF

Homework

- **Further Reading**
  - Chapter 7
- **Exercises**
  - 7.1, 7.2, 7.6
  - Any two from (7.30, 7.31, 7.32, 7.33, 7.34)
- **Submission**
  - Deadline: April 16, 2025

End of Lecture 6